Report for Project 3:

1. A high-level description of each of your public member functions in each of your classes, and why you chose to define each member function in its host class; also explain why (or why not) you decided to make each function virtual or pure virtual. For example, “I chose to define a pure virtual version of the sneeze() function in my base Actor class because all actors in Bugs! are able to sneeze, and each type of actor sneezes in a different way.”

I have followed this design, so inheritance of functions should be followed from this map

GraphObject

Actor

Pebble

EnergyHolder

Food

AntHill

Pheromone

TriggerableActor

WaterPool

Poison

Insect

Ant

Grasshopper

BabyGrasshopper

AdultGrasshopper

Class StudentWorld

void setDisplayText(); //arranges the display text in a proper format and prints it

Field\* get\_Field(); //returns the pointer to the field object

bool canMoveTo(int x, int y) const; //returns if that position is moveable(no pebbles)

StudentWorld(std::string assetDir); //constructor

virtual ~StudentWorld(); //destructor that calls cleanup()

virtual int init(); //takes information from bug files and field files to create objects

virtual int move(); //asks every object on the field to doSomething() each tick, updates the objects that moved and deletes the objects that died.

virtual void cleanUp(); //erases every object from the list

void addActor(Actor\* a); //adds an object to the field and the list

bool myHill(int x, int y, int colony); //checks if (x,y) has the colony of the ant who calls the function

Actor\* getEdibleAt(int x, int y) const; //if there is food at (x,y) return a pointer to it

Actor\* getPheromoneAt(int x, int y, int colony) const; //if there is a proper pheromone at (x,y), return a pointer to it

// Poison all poisonable actors at x,y.

bool poisonAllPoisonableAt(int x, int y);

// Stun all stunnable actors at x,y.

bool stunAllStunnableAt(int x, int y);

//basically looking for enemy ants and grasshoppers

bool isEnemyAt(int x, int y, int colony, Actor\* ptr);

//basically isEnemyAt and poison and pool

bool isDangerAt(int x, int y, int colony, Actor\* ptr) const;

//if there is an enemy on the position of (\*me), bite a random one

bool biteEnemy(Actor\* me, int biteDamage);

//Change the score, which is stored in an array.

void setScore(int colony, int amt);

//looks at the array and selects the biggest score and returns the name of the ant

std::string isWinner(int size);

//checks if the greatest score is above 6, which is the number you need to win

int isWinnerCount(int size);

class Actor : public GraphObject

{

public:

Actor(StudentWorld\* world, int startX, int startY, Direction startDir, int imageID, int depth); //constructor

StudentWorld\* getWorld() const; //the connection between StudentWorld and Actor class

virtual ~Actor(); //destructor

void setoldX(int x); //these 4 functions keep track of the position

int getoldX(); //of the actors so that if an actor moves,

void setoldY(int y); //they can reassign the position on the list

int getoldY(); //add a proper pointer to it

virtual void doSomething() = 0; //I chose to make this function pure virtual, because every actor has a different doSomething() procedure

Actor::Direction randomDir(); //returns a random direction

Basically all of the functions below are created to be used via the actor pointer. This means that they have to be defined as virtual classes for us to be able to reach the more specialized versions of these functions. It also means that we cannot use a pure virtual form of these because Actor pointer has to be able to reach it. These functions do not have a meaningful purpose in this class, but the inherited versions of these classes are functional

virtual bool blocksMovement() const; //blocks movement //for pebbles

virtual bool isDead() const; //for all energy holders

virtual void setDead();

virtual bool isBitten(); //for insects

virtual bool isInsect();

virtual bool isPoisoned();

virtual bool isStunned();

virtual void increaseStrength(); //for pheromones

virtual bool isPheromone(int colony) const;

virtual int getColony() const; //for ants, anthills and pheromones

virtual bool isAnt();

virtual bool isMyHill(int colony) const;

virtual void updateBitten(bool bitten);

// Cause this actor to be be bitten, suffering an amount of damage.

virtual void getBitten(int amt);

// Cause this actor to be be poisoned.

virtual void getPoisoned(); //to allow poison to work

// Cause this actor to be be stunned.

virtual void getStunned(); //to allow pool to work

// Adjust this actor's amount of energy upward or downward.

virtual void updateEnergy(int amt);

// Get this actor's amount of energy (for a Pheromone, same as strength).

virtual int getEnergy() const;

// Can this actor be picked up to be eaten?

virtual bool isEdible() const;

// Is this actor an enemy of an ant of the indicated colony?

virtual bool isEnemy(int colony);

// Is this actor detected as dangerous by an ant of the indicated colony?

virtual bool isDangerous() const;

// Is this actor the anthill of the indicated colony?

virtual bool isAntHill(int colony) const;

//is this an adult grasshopper

virtual bool isAdult();

/////////////////////////

//// PEBBLE /////////

/////////////////////////

class Pebble : public Actor

{

public:

Pebble(StudentWorld\* sw, int startX, int startY);

virtual void doSomething();

virtual bool blocksMovement() const;

};

///////////////////////////////

//// ENERGYHOLDER /////////

//////////////////////////////

class EnergyHolder : public Actor

{

public:

EnergyHolder(StudentWorld\* sw, int startX, int startY, Direction startDir, int energy, int imageID, int depth);

virtual bool isDead() const;

virtual void setDead();

// Get this actor's amount of energy (for a Pheromone, same as strength).

virtual int getEnergy() const;

// Adjust this actor's amount of energy upward or downward.

virtual void updateEnergy(int amt);

// Add an amount of food to this actor's location.

void addFood(int amt);

// Have this actor pick up an amount of food.

int pickupFood(int amt);

// Have this actor pick up an amount of food and eat it.

int pickupAndEatFood(int amt);

///////////////////////////////

//// FOOD /////////

//////////////////////////////

class Food : public EnergyHolder

{

public:

Food(StudentWorld\* sw, int startX, int startY, int energy);

virtual void doSomething();

virtual bool isEdible() const;

};

///////////////////////////////

//// ANTHILL /////////

//////////////////////////////

class AntHill : public EnergyHolder

{

public:

AntHill(StudentWorld\* sw, int startX, int startY, int colony, Compiler\* program);

virtual void doSomething();

virtual bool isMyHill(int colony) const;

virtual int getColony() const;

void giveBirth(); //gives birth to a new ant if it has 2000 hit points

Compiler\* getCompiler(); //returns a pointer to the compiler

};

///////////////////////////////

//// PHEROMONE /////////

//////////////////////////////

class Pheromone : public EnergyHolder

{

public:

Pheromone(StudentWorld\* sw, int startX, int startY, int colony);

virtual void doSomething();

virtual bool isPheromone(int colony) const;

virtual void increaseStrength();

};

///////////////////////////////

//// TRIGGERABLE /////////

//////////////////////////////

class TriggerableActor : public Actor

{

public:

TriggerableActor(StudentWorld\* sw, int x, int y, int imageID);

virtual bool isDangerous() const;

};

///////////////////////////////

//// WATERPOOL /////////

//////////////////////////////

class WaterPool : public TriggerableActor

{

public:

WaterPool(StudentWorld\* sw, int x, int y);

virtual void doSomething();

};

///////////////////////////////

//// POISON /////////

//////////////////////////////

class Poison : public TriggerableActor

{

public:

Poison(StudentWorld\* sw, int x, int y);

virtual void doSomething();

};

///////////////////////////////

//// INSECT /////////

//////////////////////////////

class Insect : public EnergyHolder

{

public:

Insect(StudentWorld\* world, int startX, int startY, int energy, int imageID);

virtual void doSomething();

virtual void getPoisoned();

virtual void getStunned();

virtual bool isEnemy(int colony);

virtual bool isPoisoned();

virtual bool isInsect();

virtual bool isStunned();

void setPoisoned(bool poison); //these functions are to keep track

virtual void setStunned(bool stun); //of the new private member

void setSleepCount(int count); //variables such as m\_poisoned,

int getSleepCount(); //m\_stunned and m\_sleepCount

virtual bool attemptMove(int x, int y);

virtual void getBitten(int amt);

};

///////////////////////////////

//// ANT /////////

//////////////////////////////

class Ant : public Insect

{

public:

Ant(StudentWorld\* sw, int startX, int startY, int colony, Compiler\* program, int imageID);

virtual void doSomething();

bool interpreter(); //interpreter deals with the compiler command

bool moveForwardIfPossible(); //all of them are pretty self-explanatory

bool isBitten();

void updateBitten(bool bitten);

bool isBlocked();

void updateBlocked(bool blocked);

void addPheromone();

Actor::Direction rotateCcw();

Actor::Direction rotateCw();

bool isItTrue(int operand);

bool isPheromoneAt(int x, int y, int colony);

virtual int getColony() const;

virtual bool isAnt();

virtual bool isEnemy(int colony);

};

///////////////////////////////

//// GRASSHOPPER /////////

//////////////////////////////

class Grasshopper : public Insect

{

public:

Grasshopper(StudentWorld\* sw, int startX, int startY, int energy, int imageID);

void setDesiredDistance(int distance); //both grasshoppers have

int getDesiredDistance(); //this feature so they share it

};

////////////////////////////////////

//// BABY GRASSHOPPER /////////

///////////////////////////////////

class BabyGrasshopper : public Grasshopper

{

public:

BabyGrasshopper(StudentWorld\* sw, int startX, int startY);

virtual void doSomething();

};

////////////////////////////////////

//// ADULT GRASSHOPPER /////////

///////////////////////////////////

class AdultGrasshopper : public Grasshopper

{

public:

AdultGrasshopper(StudentWorld\* sw, int startX, int startY);

virtual void getPoisoned(); //adult grasshoppers do not get poisoned or stunned, so we need to redefine those functions for this class

virtual void getStunned();

virtual void doSomething();

virtual bool isAdult();

};

1. A list of all functionality that you failed to finish as well as known bugs in your classes

When ants are in a tie at the end of the game, my program does not correctly set the winner ant. If there’s a tie, the program chooses the ant that comes before in the array (so kind of a random behavior).

In order to reach the field, I created a field with the *new* command. I tried deleting it or setting it to a nullptr in the cleanup() function; however, both gave me different errors. When I chose to do nothing, no errors were given.

1. A list of other design decisions and assumptions you made

For example, let’s say there is a pool in (x,y) that stuns every enemy on that square every turn (if they’re not already stunned) with its doSomething() function. In a turn, an ant comes to pool’s coordinates (x,y) after pool’s doSomething() function is processed. Since the pool has already completed its turn, the ant doesn’t get stunned on that tick, but instead, it gets stunned in the next tick, when the pool can actually see that there’s a non-stunned ant.

It was not very clear what to do with the pools in the spec. For example, in one place it said “make the object sleep an additional 2 turns” and in another place it said “so the object should sleep for two turns.” It was ambiguous, yet I choose to increment the sleep count by 2.

1. A description of how you tested each of your classes (1-2 paragraphs per class).

Actor class was mainly the base class of all other functional classes. In order to test it, and in order to actually complete the project, I had to create many Actor pointers and with the use of inheritance and polymorphism, my functions worked well together.

Pebble class was like a wall. No purpose, just to block other objects. All of my insects use canMoveTo and attemptMove functions, so they do not go on a square with a pebble on it. This also means that pebble class works.

Basically, all of the classes after this point use Energyholder class as a base class and they all inherit the hit point concept. This concept is the one that makes the game fun, because the objects die or adult grasshoppers born thanks to this concept.

Food class was just like a fuel tank. Food objects sacrificed themselves to feed the other objects. When food is taken away from a square, if its hit point has reached 0, it dies and is erased from the field and the list.

AntHill class is the one that is connected to compiler. It can create ants and tell the StudentWorld to increment the counter.

Pheromone class is how ants would get back to their AntHill after a long time of foraging.

TriggerableActor class is just clears the way for WaterPool and Poison by arranging little details like depth and direction. It also has the isDangerous() function which is used to find the danger in a square. We do not create pure triggerable objects

Waterpool class is just increases the stunCount by 2. Basically, the objects cannot move for 2 turns. However, the adult grasshoppers are immune

Poison class is the most powerful attacker on this game. Like a real poison, it just takes away some health from an object.

Insect class is the one with “living” objects. These objects can basically move, eat, hold food, bite, bite back...etc. It is the class where we can track all of their poison, sleep and stun states. We do not create pure insect objects.

Unlike almost every object on this game, Ant class is created in Actor class by AntHill. It is controlled by a bug file, so it is the only thing that does not have a specific behavior, which makes it hard to test. The way I did was, I forced the command to be the number that I wanted to test and it worked.

BabyGrasshopper class is the little green ones who move around and sleep all the time.

AdultGrasshopper class is an evolved version of BabyGrasshopper, it jumps, it bites, it is immune to almost everything.

Overall, the testing was the hardest for insects, because there was a lot to check. Luckily, it didn’t cause a big problem with the help of the debugger.